Universidad de Alcalá

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

DESARROLLO DE UN SISTEMA DE CONTROL DE MARCHAS PARA UN VEHÍCULO AUTÓNOMO

Autor: Pedro López Cano

Tutor/es: Ignacio Parra Alonso y Rubén Izquierdo Gonzalo

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Desarrollo de un sistema de control de marchas para un vehículo autónomo

Autor: Pedro López Cano

Tutor/es: Ignacio Parra Alonso y Rubén Izquierdo Gonzalo

TRIBUNAL:

Presidente: Melquíades Carbajo Martín

Vocal 1º: Javier de Pedro Carracedo

Vocal 2º: Ignacio Parra Alonso

FECHA: 22 de septiembre de 2017

“*A mis padres y mis hermanas, por todo su apoyo;*

A toda la familia, en especial a los primos;

A todos mis amigos, los viejos y los nuevos, por todos los momentos compartidos;

A los que han aparecido por sorpresa;

A los compañeros de AIRIS;

A David y Agustín, por estar siempre dispuestos a ayudar;

A ti, aunque ya no compartamos el camino.

Y por último, pero no menos importante,

A Rubén, por su disponibilidad absoluta.”

“Los científicos estudian el mundo tal como es;

Los ingenieros crean el mundo que nunca ha sido.”

Theodore Von Karman (Físico húngaro-estadounidense)

Índice general

Índice de figuras

I. RESUMEN

En este documento se detalla el diseño de un sistema de control de marchas para un vehículo autónomo. Este sistema se encarga de generar las señales necesarias para cambiar entre modo Drive y modo Manual, y en modo Manual, las necesarias para subir y bajar de marcha. También se explica el proceso de creación de una placa de circuito impreso para su montaje. Por último, se desarrollará un algoritmo para su integración en el vehículo autónomo del grupo INVETT.

Palabras clave: Sistema de control de marchas, ECU, conducción autónoma, INVETT.

II. ABSTRACT

This document details the design of a gear control system for an autonomous vehicle. This system is responsible for generating the signals necessary to switch between Drive mode and Manual mode, and in Manual mode, those necessary to go up and down. It also explains the process of creating a printed circuit board for assembly. Finally, an algorithm for its integration into the autonomous vehicle of the INVETT group will be developed.

Keywords: Gear control system, ECU, autonomous driving, INVETT.

III. RESUMEN EXTENDIDO

Actualmente el tema de la conducción autónoma está en auge y es el futuro de la conducción, pero para ello, es necesario que los vehículos sean capaces de tomar decisiones por sí mismos.

Para la investigación en este campo, es necesario adaptar vehículos dotándolos de herramientas hardware y software que les permitan percibir el medio que los rodea y poder tomar decisiones. El grupo de investigación INVETT ha dotado a un Citröen C4 de multitud de sensores y ha desarrollado un conjunto de algoritmos, que permiten al vehículo circular de manera autónoma. Además, es capaz de comunicarse con otros vehículos para circular de forma cooperativa.

En este TFG (Trabajo Fin de Grado) se diseñará un sistema de control para los cambios del vehículo del grupo INVETT.

El proceso de desarrollo del sistema de control se ha desarrollado en 2 fases. La primera ha consistido en un desarrollo del hardware necesario para la generación de las señales de control. La segunda fase se ha centrado en el desarrollo del software necesario para controlar el sistema diseñado anteriormente.

En cuanto al desarrollo del hardware, se empezó por realizar un estudio del vehículo para conocer el funcionamiento del sistema de cambios. Para ello, se desmotó la palanca de cambios para conocer si era un sistema electrónico, o bien, un sistema mecánico. Tras este primer reconocimiento, se ha llegado a la conclusión de que es un sistema electrónico consistente en un sensor en la palanca de cambios que envía señales a la unidad de control de motor (ECU).

El siguiente paso, es conocer la lógica por la que se rige la ECU y diseñar un sistema capaz de replicar esta lógica. Para esto, se ha optado por un microcontrolador y una tarjeta de circuito impreso en la que implementar el circuito electrónico necesario. Este circuito, además de replicar las señales, será capaz de leer el estado de la palanca

de cambios y mostrar de una manera sencilla y visual el estado del sistema mediante leds y una pantalla LCD.

La segunda parte del proceso de desarrollo ha consistido en la programación del software necesario para controlar el circuito electrónico. Se han utilizado dos estilos de programación diferentes, uno para la inicialización del sistema y otro para la generación de las señales. El primero es una máquina de estados, que además de inicializar el sistema, sirve como puerta de acceso para el PC que controla el vehículo, pues es necesario recibir un ID para poder pasar a la generación de señales.

El proceso de generación de señales se ha llevado a cabo mediante un sistema de procesos. Consiste en una matriz de tareas por hacer y una interrupción temporizada que las ejecuta en el momento necesario. Se ha creado un conjunto de instrucciones para gestionar esta matriz, es decir, añadir tareas o eliminarlas.

También se ha desarrollado un algoritmo para reducir el error y las distorsiones cuando el sistema de control del vehículo necesita seguir una referencia de velocidad. Estas desviaciones son debidas a cambios de marcha. Por tanto, el algoritmo desarrollado trata de evitar cambios de marcha innecesarios.

IV. MEMORIA

Capítulo 1º
Introducción

En este capítulo se detallarán tanto las motivaciones que han impulsado la realización de este proyecto, como los objetivos perseguidos con su desarrollo. También se hará un esquema que explique los contenidos de los diferentes apartados que forman esta memoria.

1.1. Motivación

Se entiende por vehículo autónomo aquel que es capaz de circular sin la necesidad de que una persona actúe para controlarlo. El vehículo es capaz de percibir el medio que lo rodea y tomar las decisiones necesarias. Es justamente, en el apartado de tomar decisiones, donde entra en juego el sistema de control desarrollado en este proyecto.

Dentro de la conducción autónoma, el grupo INVETT [1] de la Universidad de Alcalá ha desarrollado numerosos algoritmos de conducción cooperativa. La motivación para la realización de este proyecto surge tras la competición internacional de vehículos autónomos del GCDC [2] (Grand Cooperative Driving Challenge) del año 2016, en la que participó el grupo INVETT.

Figura 1: Vehículo del grupo INVEET

Una de las pruebas o escenarios del GCDC 2016 consistía en la simulación de una zona de obras en una autopista. En esta prueba dos pelotones de vehículos debían sincronizarse con el fin de conseguir fusionarse en uno solo y atravesar la zona en obras por un solo carril. Es necesaria una perfecta sincronización entre los vehículos ya que si un vehículo se bloquea el resto también lo hará. Para realizar esta sincronización, se debe seguir una referencia de velocidad o guardar una distancia con el vehículo que se encuentra delante. Antes de la realización e implementación del sistema de control de marchas de este proyecto, el vehículo autónomo del grupo INVETT era capaz de actuar sobre la dirección y los pedales de aceleración y freno.

Como la caja de cambios es automática, el vehículo cambiaba cuando creía que era necesario. Esto provocaba distorsiones en la referencia de velocidad que a su vez se traducían en errores en la distancia de seguimiento.

El sistema de control del vehículo, a través del sistema desarrollado en este proyecto, podrá gestionar de forma inteligente los cambios, evitando que estos se produzcan cuando no sea necesario con el fin de generar perfiles de velocidad más precisos.

1.2. Objetivos

El principal objetivo es el de diseñar un sistema capaz de generar las señales de control que son enviadas a la unidad de control de motor o ECU (del inglés: Engine Control Unit) del coche, para que sea este sistema de control el encargado de enviar las señales adecuadas a los requerimientos y, por tanto, seleccionar la marcha apropiada. Además, el sistema de control debe poder desconectarse de manera rápida en caso de anomalía para devolver el control al usuario sin que esto afecte a la seguridad.

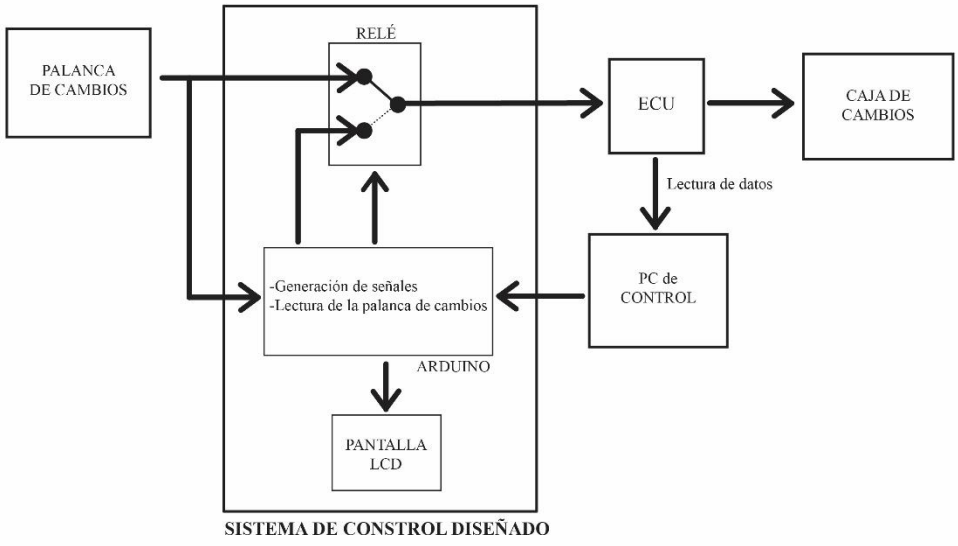


Figura 2: Diagrama de bloques del sistema

El sistema de control se programará en una placa controladora comercial del fabricante japonés XCSource [3]. Esta placa está basada en la plataforma de código abierto Arduino [4] y más concretamente en el modelo Arduino Nano [5]. Se ha optado por este microcontrolador por su fácil disponibilidad y por proporcionar un entorno de desarrollo (Arduino IDE [6]) gratuito y muy potente que facilitan su uso en proyectos de cualquier índole.

Además del microcontrolador, será necesaria una placa de circuito impreso o PCB (del inglés: Printed Circuit Board) sobre la que montar el hardware requerido en el diseño del sistema de control. Para el diseño de esta PCB se usará el software libre DesignSpark PCB [7].

Por último, se desarrollará un algoritmo que gestione el cambio de marcha en función del estado del vehículo y las órdenes o consignas generadas por él.

1.3. Esquema de la memoria

A continuación se describen brevemente los diferentes apartados que forman esta memoria:

- **Capítulo 1. Introducción.** Es el capítulo actual y se trata de una introducción al trabajo realizado en este proyecto.
- **Capítulo 2. Estudio del hardware del vehículo.** En este capítulo se describe el funcionamiento del hardware que controla el sistema de cambios del vehículo.
- **Capítulo 3. Diseño hardware del sistema de control.** Se explica tanto el hardware diseñado como el proceso de creación de la placa de circuito impreso. En la parte de diseño se detallarán los cálculos realizados y en la parte de creación se explicarán los pasos realizados en la plataforma de desarrollo.
- **Capítulo 4. Lógica de control para el sistema hardware.** Este capítulo trata principalmente de explicar y detallar el software desarrollado para controlar el hardware que se ha diseñado en el capítulo anterior. Además se describirá el algoritmo que integrará el sistema de control diseñado con el sistema de control propio del vehículo autónomo.
- **Capítulo 5. Resultados.** En este capítulo se muestra el hardware final y exponen los resultados obtenidos tras los ensayos de campo.
- **Capítulo 6. Conclusiones y trabajos futuros.** Se exponen las conclusiones obtenidas tras la realización de este proyecto se proponen trabajos futuros.

Capítulo 2º
Estudio del hardware del vehículo

En este capítulo se describe el hardware encargado del sistema de cambios del vehículo. Además, se realiza un estudio del mismo con el fin de entender su funcionamiento para poder replicarlo a través del microcontrolador.

2.1. Palanca de cambios

El vehículo para el que se va a realizar el sistema de control de marchas, es el Citroën C4 del grupo INVETT. Las principales características del vehículo se pueden ver en la tabla 1.

Modelo	**Citroën C4 5p 1.6i 16v VTR PLUS**
Potencia máxima CV - kW / rpm	**109 - 80 / 5750**
Par máximo Nm / rpm	**147 / 4000**
Caja de cambios	**Automática, 4 velocidades**
Embrague	**Centrifugo de fluido**
Peso (kg)	**1349**
Largo / Ancho / Alto (mm)	**4260 / 1773 / 1471**

Tabla 1: Características del vehículo

Como se puede ver en la tabla 1, el sistema de cambios es automático con 4 velocidades. La palanca de cambios consta de 5 posiciones: Parking, se usa únicamente cuando el vehículo está estacionado; Reverse, sirve para ir marcha atrás; Neutral, punto muerto, es decir, no está engranada ninguna marcha; Drive, es el modo automático, por tanto es la ECU del coche la que decide que marcha se ha de poner en cada momento; Manual, en esta posición el vehículo permanecerá en la misma marcha hasta que el usuario indique lo contrario empujando la palanca hacia arriba, o hacia abajo en función de si quiere subir, o bajar marcha.

Figura 3: Palanca de cambios

Tras desmontar la palanca de cambios, se comprueba que los sensores que detectan que la palanca está en las posiciones Parking, Reverse y Neutral se encuentran dentro del compartimento del motor. Las posiciones Drive y Manual se indican a través de un sensor situado bajo la propia palanca. Ante la imposibilidad de acceder al resto de sensores, se decidió replicar únicamente las posiciones Drive y Manual, ya que esto es suficiente para tener control total del vehículo en la dirección de avance.

2.2. Sensor de selección entre Drive y Manual

El circuito encargado de comunicar a la ECU en qué posición está la palanca de cambios es el que podemos ver en la figura 2.

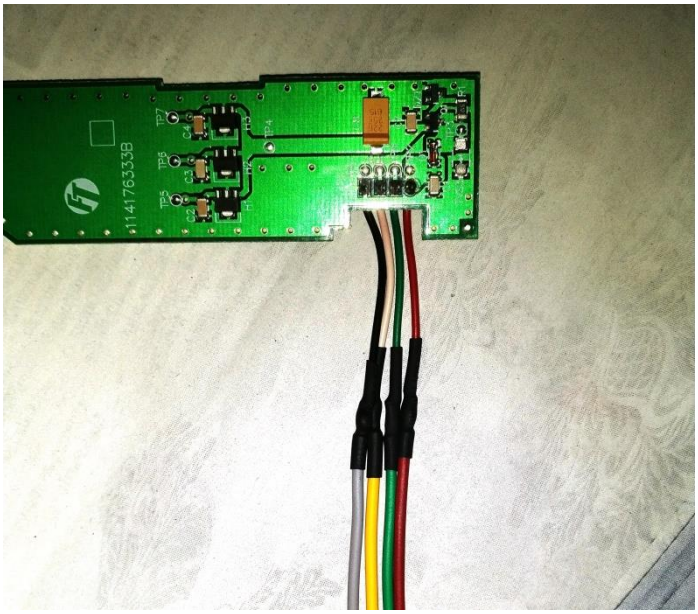


Figura 4: Sensor de la palanca de cambios

Como se puede ver en la figura 2, hay 4 cables conectados al sensor. Para conocer su función se ha medido la tensión de los mismos en las diferentes posiciones de la palanca de cambios. A continuación, se muestra una tabla con la información obtenida, entendiendo por "alto" un nivel de tensión superior a 12V y "bajo" un nivel inferior a 2V.

		COLOR DEL CABLE			
POSICION DE LA PALANCA		**GRIS**	**AMARILLO**	**VERDE**	**ROJO**
	DRIVE	Bajo	Alto	Alto	Alto
	MANUAL	Bajo	Bajo	Bajo	Alto
	SUBIR MARCHA	Bajo	Alto	Bajo	Alto
	BAJAR MARCHA	Bajo	Bajo	Alto	Alto

Tabla 2: Niveles de tensión en los cables del sensor

Cabe destacar que cuando se lleva la palanca de cambios a la posición subir marcha o bajar marcha, esta vuelve automáticamente a la posición secuencial, por lo que cuando se quiere subir o bajar marcha realmente hay que enviar un pulso a la ECU en la señal correspondiente. Se ha comprobado que el tiempo mínimo para que la unidad de control de motor entienda este pulso es de 50ms.

A partir de los datos de la tabla 2, se puede deducir que los cables Rojo y Gris son la alimentación del sensor, a partir de ahora VCC y GND respectivamente y que los cables de color Amarillo y Verde son las señales de control para la ECU, S1 y S2 respectivamente en el resto de la memoria.

Capítulo 3º
Diseño hardware del sistema de control

En este capítulo se explica tanto el funcionamiento del hardware diseñado, como el proceso de creación de la tarjeta de circuito impreso necesaria para su implementación. La función del circuito es generar las señales de control y gestionar si es el propio sensor de la palanca de cambios o el microcontrolador quien envía estas señales a la ECU. El proceso de conmutación entre un origen y otro se hace a través de un relé.

3.1. Esquema de conexiones

A continuación se muestra el esquema de conexiones del circuito electrónico diseñado. Se puede ver un esquema con mayor resolución en el anexo I. En el esquema se pueden distinguir 3 partes claramente diferenciadas:

- Parte 1: Circuito de indicadores luminosos.
- Parte 2: Circuito para la gestión de las señales.
- Parte 3: Circuito de lectura del estado de la palanca de cambios.

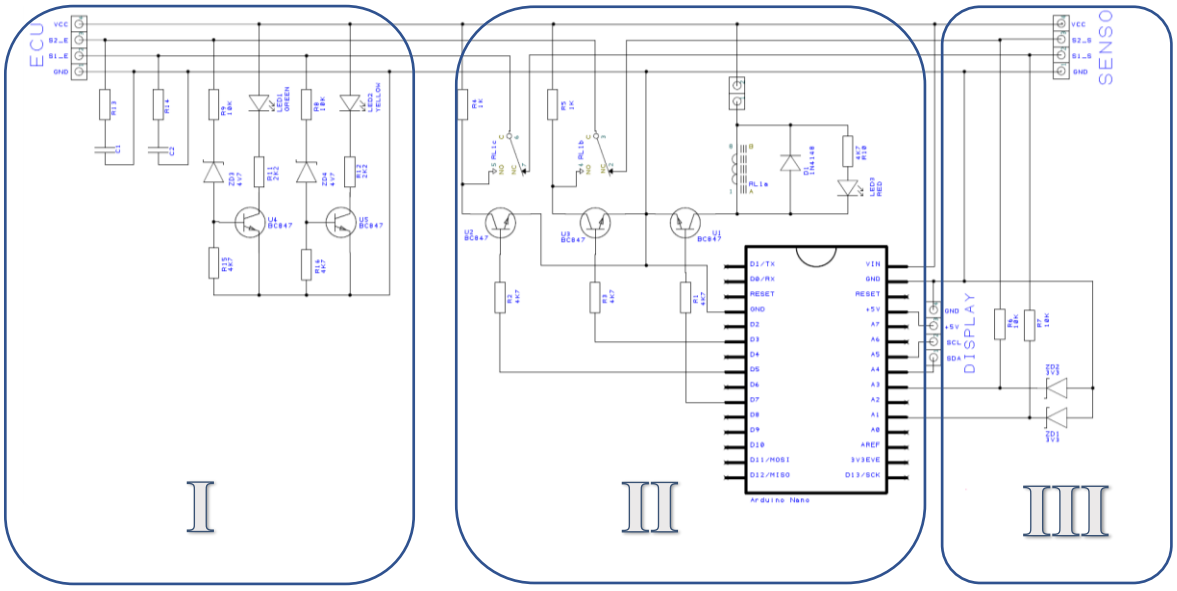


Figura 5: Esquema de conexiones

3.2. Parte 1: Circuito de indicadores luminosos

El primer sub-circuito, ver figura 4, consta de 2 diodos led conectados entre VCC y el colector de un transistor bipolar BC847. Este transistor se encarga de conectar o desconectar los diodos led ya que su emisor está conectado a GND. La activación de los transistores está vinculada a la tensión de las señales S1 y S2 respectivamente. Dado que el circuito para ambas señales es el mismo, se explicará solamente el cálculo de una de las señales.

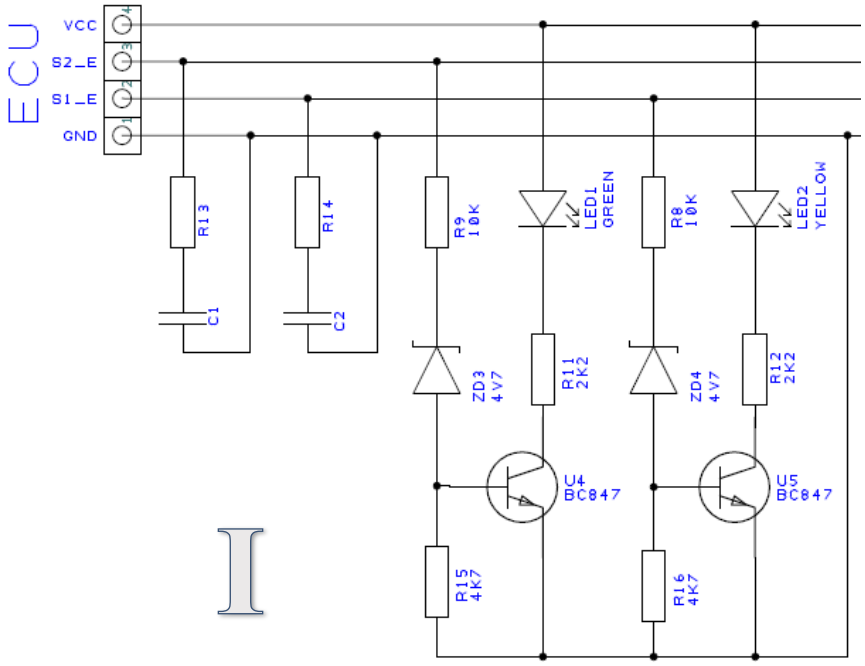


Figura 6: Sub-circuito 1

Para que se active el transistor U4, es necesaria una tensión $V_{be} > V_{besat}$. Estudiando el circuito llegamos a la siguiente igualdad:

$$V_{be} = V_S - V_{R9} - V_Z \ (1)$$

$$I_D = \frac{V_s - V_Z}{R_9 + R_{15} + R_z} (2)$$

Donde V_s es la tensión presente en S2, V_{R9} la caída de tensión en R9, V_Z es la tensión zener del diodo DZ3 e I_D la corriente por el diodo. La función del diodo zener es la de garantizar que la conexión del transistor solo se produzca cuando haya una tensión en la señal S2 tal que:

$$V_{S2} > V_{be} + V_{R9} + V_Z \ (3)$$

De la ecuación 2, suponiendo VCC 12V (el peor de los casos) y despreciando el valor de R_z puesto que es mucho menor que R9 y R15 se puede obtener que I_D = 0.5mA. De la ecuación 1, sabiendo que DZ3 es un diodo MM3Z4V7B con V_z = 4,7V se obtiene que V_{be} = 2.3V. Con este valor, U4 se activará y por tanto, el LED1 podrá conducir y así indicar que hay un nivel alto en S2.

Por otro lado, los condensadores C1 y C2 y las resistencias R13 y R14 están pensados para mantener la tensión existente en las señales S1 y S2 en el momento de la conmutación del relé. En las pruebas realizadas en el vehículo, se ha comprobado que el tiempo de vuelo del relé es lo suficientemente pequeño para no influir en la ECU, por tanto, esta parte del circuito no es necesaria.

3.3. Parte 2: Circuito para la gestión de las señales

Este circuito es la parte más importante de todo el sistema hardware. Tiene 2 funciones: 1. Conmutar el relé para seleccionar el origen de las señales que se envían a la ECU; 2. Generar las señales de control.

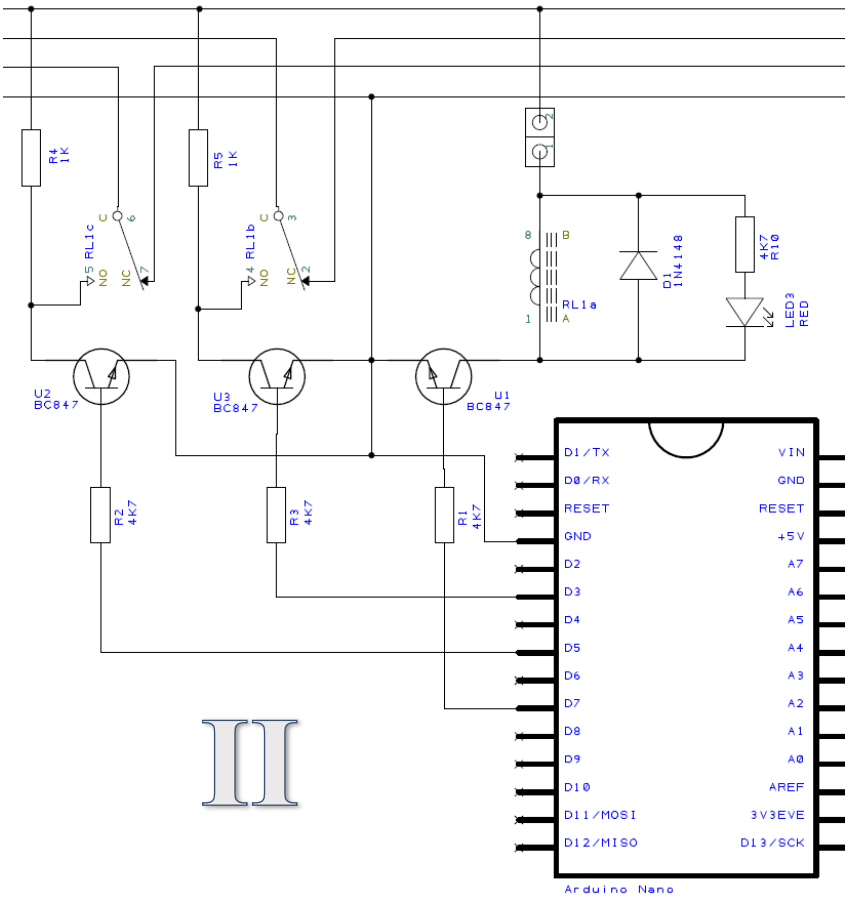


Figura 7: Sub-circuito 2

El relé es del modelo 40.52 de 12V con 2 contactos. La bobina está conectada entre VCC y el colector un transistor bipolar BC847. En la rama que se dirige a VCC (*B)*, se ha colocado un conector para incluir un interruptor que podrá, en caso de emergencia, interrumpir la alimentación de la bobina. De esta manera, los contactos del relé volverán a la posición *normalmente conectado (NC)* en el caso de que estuvieran en la posición *normalmente abierto (NO)*. En paralelo con la bobina, se ha colocado un diodo 1N4148 de forma que cuando la bobina haya almacenado energía a causa de una activación y se desconecte, la corriente tenga un lazo de descarga. Además del 1N4148, se puede observar un diodo led con su resistencia de polarización. Este led se iluminará cuando la bobina esté excitada, y así se podrá conocer el estado de la misma de manera visual. El transistor conectado al extremo de la bobina que no se dirige a VCC (*A*), está gobernado por el microcontrolador, de manera que cuando se active la salida correspondiente, se cerrará el lazo de corriente y por tanto se hará conmutar el relé.

Los dos contactos comunes (*C*) se conectarán a los cables de las señales S1 y S2 provenientes de la ECU.

Los contactos *NC1* y *NC2* del relé están conectados directamente a las señales del sensor de la palanca de cambios. Esto quiere decir que en la posición de reposo, son estas señales las que se envían a la ECU.

Cada uno de los contactos *NO1* y *NO2* está unido al colector de un transistor BC847 y a VCC a través de una resistencia. La base de dicho transistor está conectada a una salida de microcontrolador a través de una resistencia de polarización para limitar la corriente de base a $\frac{5V\text{-}V_{be}}{4K7\Omega} = 1mA$, de manera que cuando en esta salida haya un nivel bajo, en el contacto *NOx* habrá un nivel alto (12V). Cuando en la salida del microcontrolador haya un nivel alto, el transistor conducirá y por tanto el potencial del contacto *NOx* será 0V y la ECU entenderá un nivel bajo.

3.4. Parte 3: Circuito de lectura del estado de la palanca

Esta parte del circuito está pensada para conocer el estado de la palanca de cambios a través de la lectura de las señales que genera el sensor. Cuando en alguna de las señales S1 o S2 haya un nivel bajo (menor de 2V), la tensión en la entrada del microcontrolador será siempre menor que la tensión del diodo zener. De manera contraria, cuando haya un nivel alto (12V o más), la entrada del microcontrolador tendrá un potencial fijo igual a V_z. Dado que el diodo empleado es un MM3Z3V3B, la tensión que detectará el microcontrolador es de 3,3V. Con estos datos se podrá conocer la posición de la palanca de cambios.

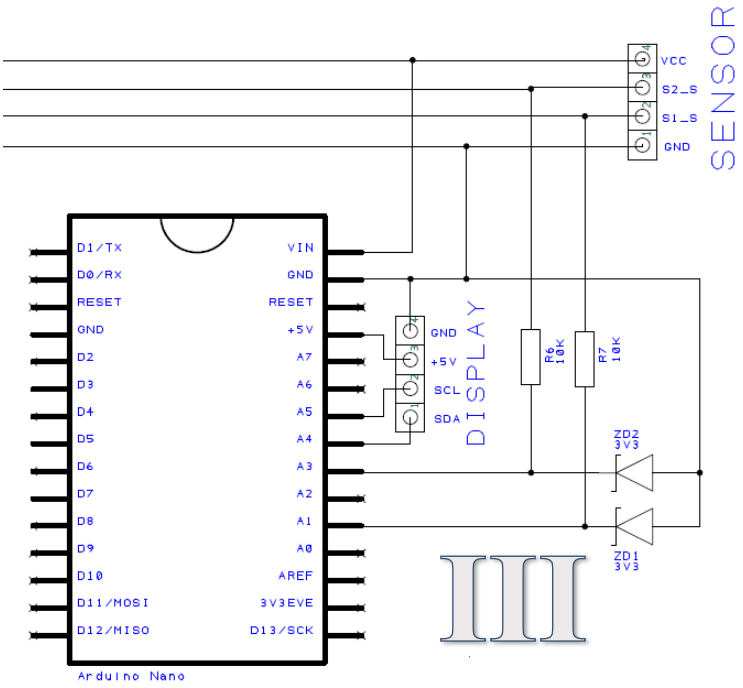


Figura 8: Sub-circuito 3

Además se ha instalado un conector para incorporar una pantalla LCD en la que se mostrará información del estado del sistema de control, de manera que el usuario pueda tener información a simple vista.

3.5. Diseño del circuito impreso

Para el diseño del circuito, se ha usado DesignSpark PCB 8.0. A continuación se va a explicar el proceso de creación de la PCB y la generación de los archivos necesarios para su fabricación.

El primer paso al abrir el software de diseño es crear un proyecto. Para ello se debe hacer clic en “File” y a continuación en “New”. Se abrirá una ventana como la que se puede ver en la figura 7. En dicha ventana, es necesario marcar “Project” y seleccionar la ruta del directorio donde se guardará el proyecto.

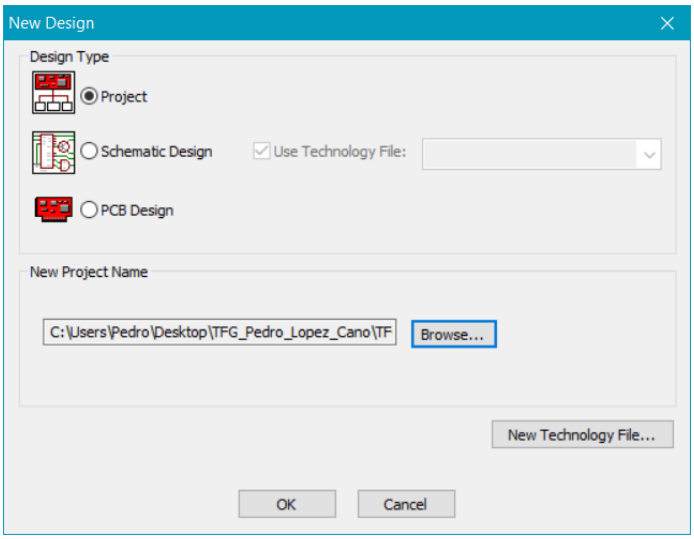


Figura 9: Nuevo proyecto

Una vez hecho esto hay que volver a hacer clic en "File" y a continuación en "New". Esta vez se debe seleccionar "Schematic Design" y marcar la casilla "Add To Open Project".

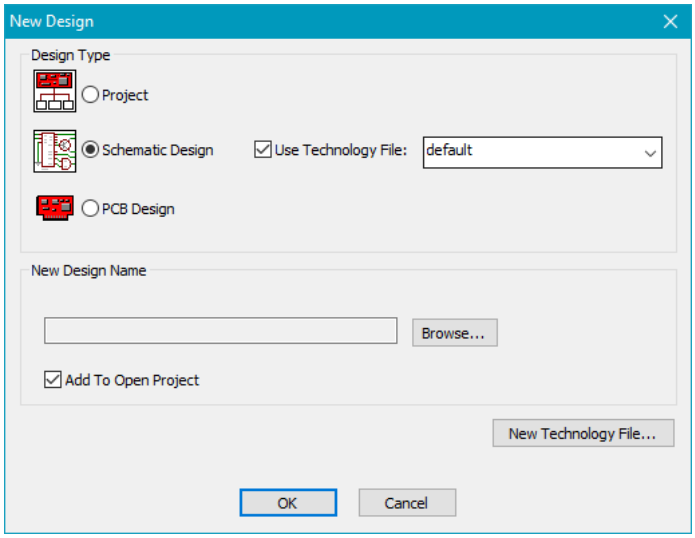


Figura 10: Nuevo esquema

A continuación, se debe empezar a crear el diseño del circuito que se quiere diseñar. Para añadir elementos nuevos hay que hacer clic en "Add" y posteriormente en "Component".

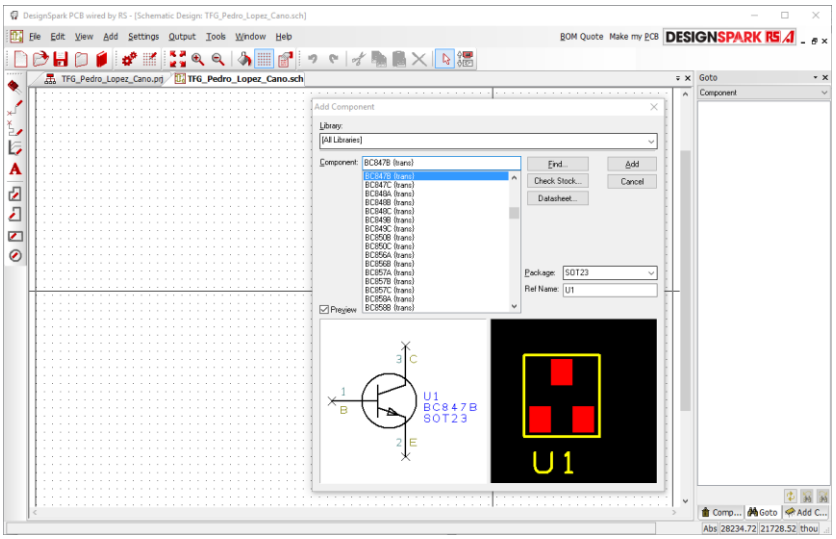


Figura 11: Añadir componente

Una vez finalizado el diseño del circuito, procedemos a su conversión a PCB, para ello, hacemos clic en “Tool” y posteriormente a “Translate to PCB”.

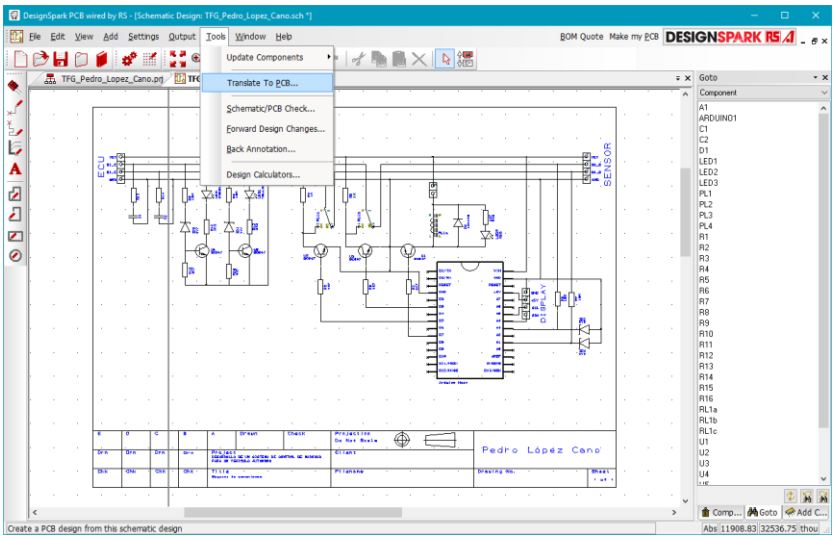


Figura 12: Conversión a PCB

Al hacer esto, se abrirá un asistente que ayudará en el proceso de creación de la placa de circuito impreso. Este asistente es importante porque define el tipo de PCB que se va a usar, el número de capas, la forma y tamaño de la placa entre otras cosas.

En la figura 11 se observa la configuración usada para la creación de este proyecto. Se ha optado por una placa rectangular con 2 capas.

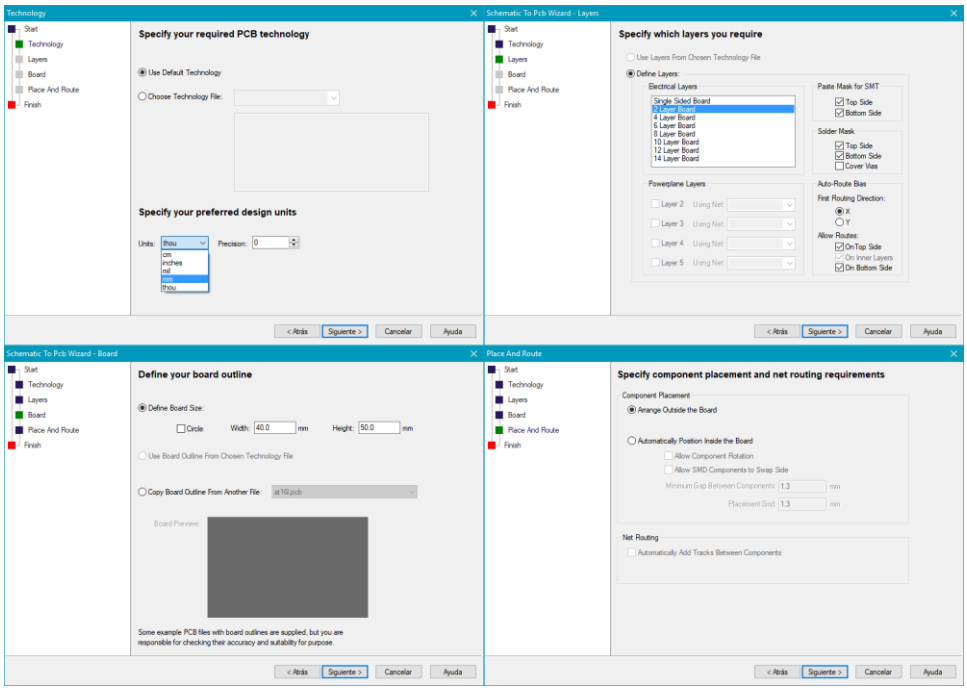


Figura 13: Asistente para la creación de la PCB

Tras finalizar el asistente, se mostrará una pantalla con los componentes usados en el esquema del proyecto colocados fuera de la PCB. Hay que colocarlos dentro del contorno del circuito impreso. Es posible cambiarlos de capa, para ello, para ello se usa la tecla “F” del teclado. Las líneas amarillas, indican conexiones eléctricas presentes en el esquema, y a medida que se vayan creando pistas, estas irán desapareciendo.

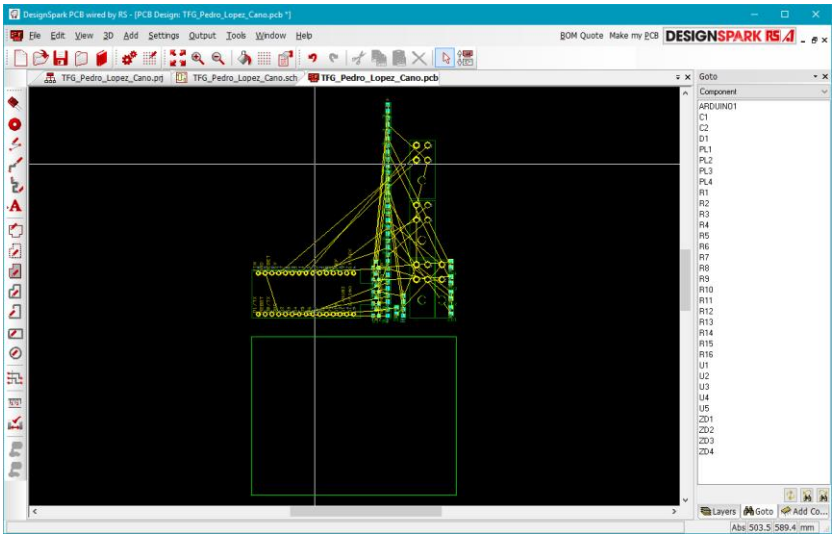


Figura 14: Colocación de componentes

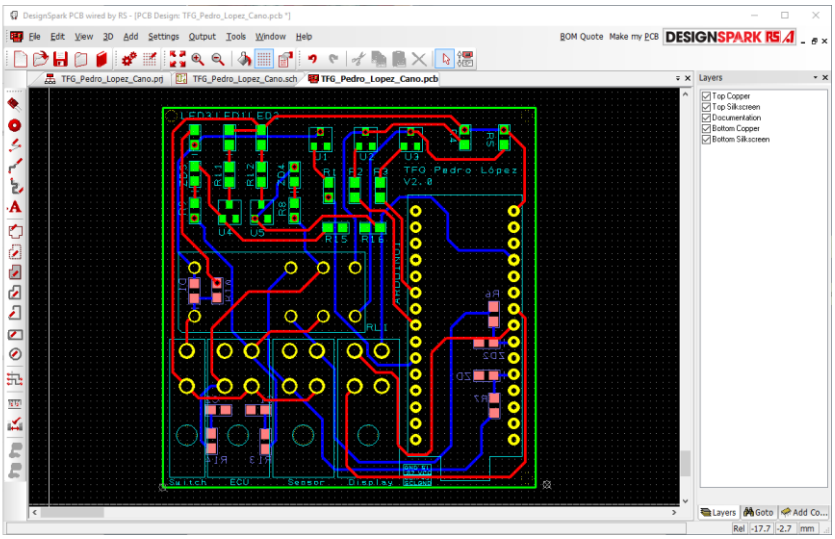


Figura 15: Diseño final

Una vez que todos los componentes están en su posición definitiva y se han creado todas las conexiones, se procede a la creación de los planos para la fabricación. Para ello, se hace clic en "Output" y posteriormente en "Manufacturing Plots".

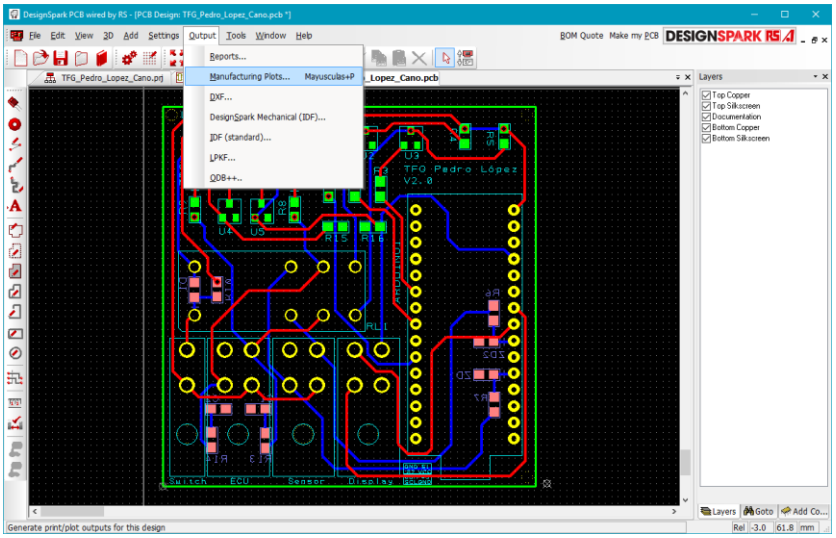


Figura 16: Creación de planos para fabricación

En la pantalla de creación de planos de fabricación es importante asegurarse que todas las capas están seleccionadas. Además, hay que crear un plano nuevo en el que se incluya el contorno de la placa del circuito impreso. Dependiendo del fabricante se deberá usar un tipo de plano u otro. En este caso, el fabricante trabaja con el formato Gerber, por tanto este es el formato seleccionado. Una vez finalizada la configuración, se hace clic en "Run". En "Options" se puede elegir el directorio para guardar los planos.

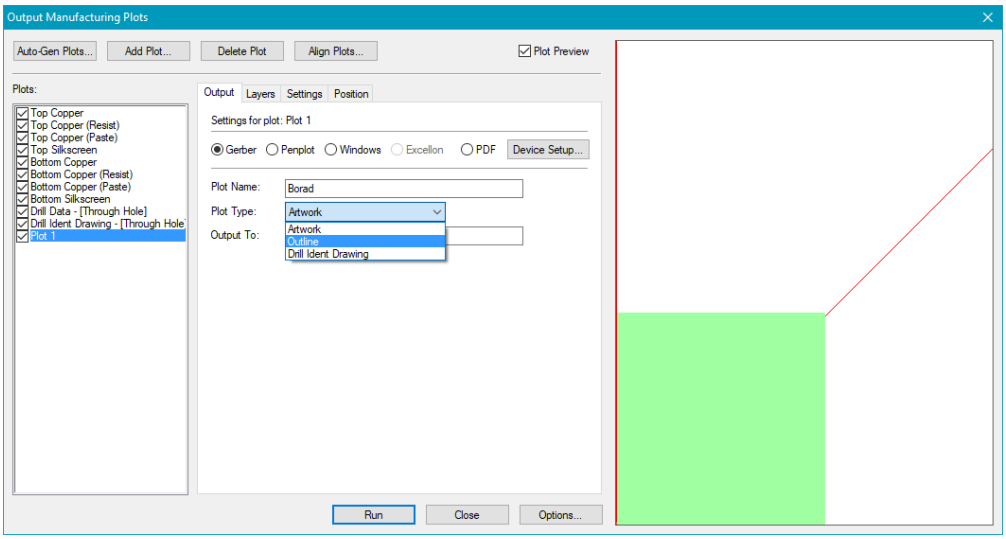


Figura 17: Creación del Gerber del contorno

Estos planos generados, son los que se han enviado al fabricante para la creación del circuito impreso. En la figura 16 se puede ver la PCB fabricada.

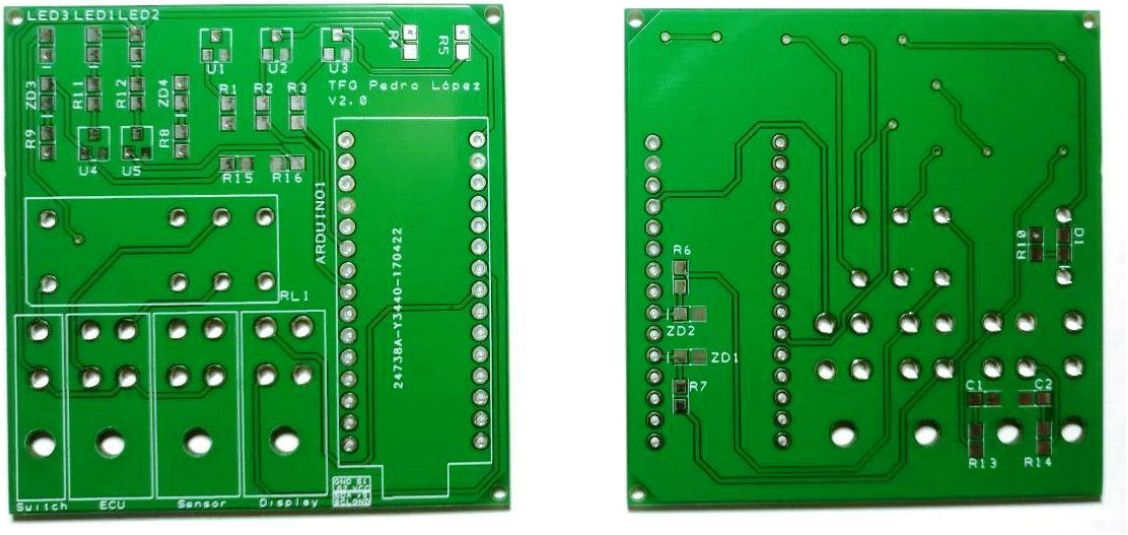


Figura 18: PCB fabricada

Capítulo 4º
Lógica de control para el sistema hardware

En este capítulo describe la lógica que se ha seguido para el control del hardware que se ha diseñado en el capítulo anterior. Se explicará el software desarrollado para que el microcontrolador genere las señales necesarias y poder probar el sistema de control.

La programación de un Arduino consta de 2 partes principales: la configuración y el bucle principal. La parte de configuración solo se ejecuta la primera vez que se enciende el microcontrolador, y es utilizada para realizar la inicialización de las entradas y salidas, así como cualquier otro parámetro que necesite ser configurado. En el bucle principal se introducen las instrucciones que deba ejecutar el microcontrolador de manera sistemática.

En este proyecto, se ha desarrollado un sistema de procesos capad de gestionar las tareas a realizar. Este sistema puede ejecutar tareas o programarlas para más adelante, puede añadir tareas a la cola o eliminar tareas que ya no se necesite ejecutar.

4.1. Función de configuración: *void setup()*

En el código fuente que se adjunta en el Anexo II, se puede encontrar la función *void setup()*. En este proyecto, esta función tiene un doble propósito, por un lado debe realizar la configuración necesaria para el control del hardware, pero por otro lado debe servir como control de acceso a la parte del programa encargada de la generación y gestión de las señales de control que serán enviadas a la ECU. Esto es necesario para poder decidir en qué momento empezar a realizar el control.

En la parte de la configuración, se definen los pines del microcontrolador que serán entradas y aquellos que serán salidas, se inicializa el puerto serie para las comunicaciones con el ordenador, se inicializa la pantalla LCD y cuando se decida pasar a controlar el sistema de cambios, se iniciará el temporizador que pone en marcha el sistema de procesos.

Para realizar el control de acceso se ha desarrollado una máquina de estados como la que se puede ver en la figura 17. De esta forma, hasta que el ID correcto no haya sido recibido por el microcontrolador, no se continuará con la ejecución del programa. Por tanto, mientras que el estado sea distinto de 3, se estará comprobando si ha llegado información al puerto serie. Si llega, se comprueba si coincide con el ID del microcontrolador y en función de la comprobación se hace una acción u otra.

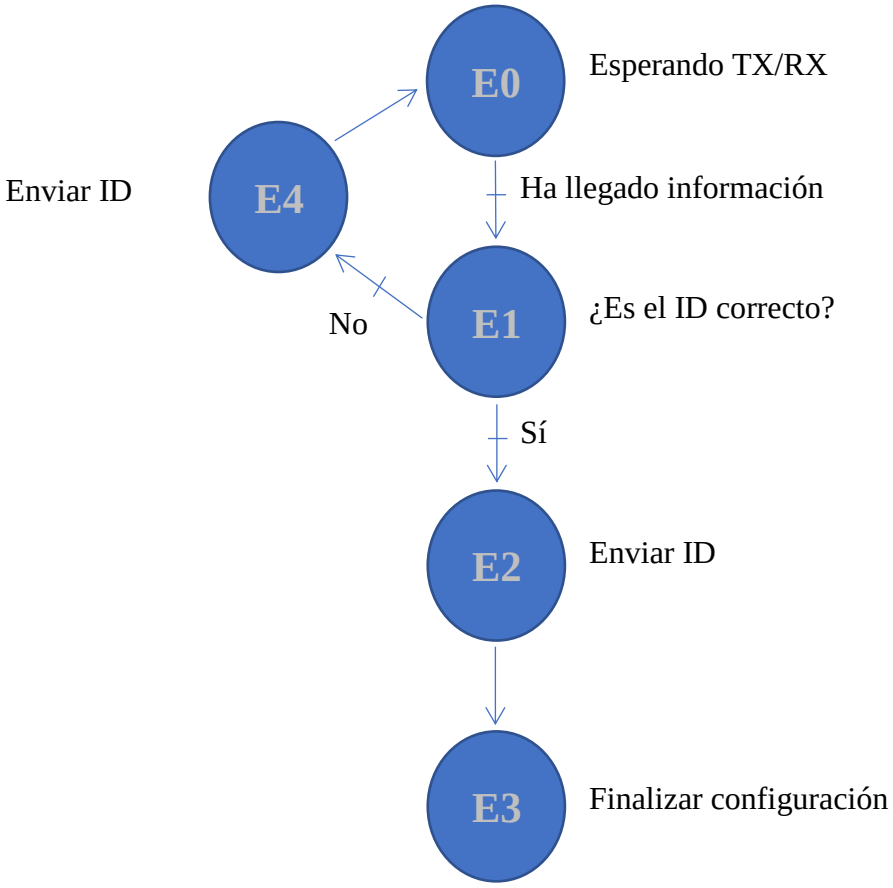


Figura 19: Diagrama de estados

4.2. Bucle principal: *void loop()*

El bucle principal, o *void loop()*, es la función principal. Las instrucciones que se encuentran en esta función se ejecutan de manera cíclica. En el presente sistema de control el bucle principal se encarga de:

1. Comprobar si se ha recibido información por el puerto serie y, en el caso de haberla recibido, evaluar su información y añadir las tareas oportunas al sistema de procesos;
2. Comprobar si ha habido algún cambio en la posición de la palanca de cambios;
3. Actualizar el estado de la pantalla LCD cuando sea necesario.

4.3. Sistema de procesos

Este sistema es el encargado de gestionar que tareas se deben realizar en cada momento. Su funcionamiento es el siguiente: se programa una interrupción cada 25ms y en la rutina de atención a la interrupción *void ISR_System()*, se comprueba si hay alguna tarea que realizar en ese momento. En caso afirmativo, se ejecuta y se incrementa el contador que indica el instante actual. En caso negativo, es decir, no hay ninguna tarea pendiente en ese instante, solamente se incrementa el contador.

Para la gestión de las tareas pendientes, se ha creado una matriz en la que se guarda la tarea a realizar y el momento en el que debe ser realizada.

Se han desarrollado 2 funciones auxiliares que permiten añadir y eliminar tareas al sistema de procesos. La función para añadir tareas, *int addTask(unsigned int t, unsigned int (*p)(void))*, recibe el instante en el que se tiene que ejecutar la tarea y un puntero a la tarea a ejecutar. Recorre la matriz de tareas hasta que encuentra una posición libre y vuelca la información. La función para eliminar tareas, *int deleteTask(unsigned int (*p)(void))*, funciona de manera similar a la anterior, pero busca la tarea que recibe dentro de la matriz de tareas. Si la encuentra, la elimina para que no se ejecute.

Además de estas 2 funciones, se ha creado otra, que elimina todas las tareas, *void deleteTasks()*.Esta función directamente borra todo el contenido de la matriz de tareas. Se usa en la configuración inicial para inicializar la matriz.

4.4. Funciones para la generación de las señales

La generación de la señales para la ECU se hace a través de 5 funciones. Cada una de ellas, activa las salidas del microcontrolador necesarias. Estas funciones son las siguientes.

La primera es: *int control_ECU()*. Esta función sirve para devolver el control al sensor, y por tanto, a la ECU. No tiene ninguna restricción para su ejecución, por lo que únicamente pone las señales S1 y S2 a nivel bajo, desconecta el relé, indica que es necesario actualizar el LCD y actualiza el estado en el que se encuentra el sistema.

A continuación se encuentra la función que simula la posición Drive de la palanca de cambios. Esta función es: *int control_MCU_DRV()*. Para activar las salidas oportunas, primero se comprueba si puede hacerse, es decir, se comprueba que el sistema de control está libre, pues podría estar subiendo o bajando marcha. Si el sistema está libre, se indica que es necesario actualizar el LCD, se hace conmutar el relé, se actúa sobre las salidas, para ello, pone un nivel bajo en ambas, lo que se traduce como un nivel alto en S1 y S2, y por último, se actualiza el estado del sistema.

El funcionamiento de *int control_MCU_SEQ()*es el mismo que la función anterior salvo que, en este algoritmo, las salidas necesarias para simular el estado Manual se configuran a nivel alto para obtener un nivel bajo en S1 y S2. La salida del relé también estará activa para realizar la conmutación.

Las funciones *int gear_up()* e *int gear_down()* tienen la misma lógica, pero una sirve para subir marcha y la otra para bajar. Primero se comprueba si el sistema está en modo Manual. Si se cumple esta condición, se configura la salida oportuna a nivel bajo para que la señal S1 o S2, dependiendo de si se quiere subir o bajar marcha, se fije en nivel alto, se indica al sistema que se está cambiando de marcha, y se programa otra llamada a esta función dentro de 50ms. En esta segunda llamada, se devuelve el sistema al modo Manual.

4.5. Algoritmo de control del sistema de cambio de marcha

En este apartado, se muestra el algoritmo que se ha desarrollado para integrar el sistema de control descrito en este proyecto con el sistema del grupo INVETT. Este algoritmo no se incluye en el conjunto de algoritmos del PC de control. Su objetivo es evitar cambios de marcha innecesarios, con el fin de generar perfiles de velocidad más precisos y con menor distorsión.

Para la lógica del sistema de control se han tenido en cuenta las siguientes restricciones para los rangos de velocidad de cada marcha:

- Primera: 0 – 40 km/h
- Segunda: 20 – 80 km/h
- Tercera: 40 – 100 km/h
- Cuarta: 70 – 120 km/h

El algoritmo tendrá como variables de entrada:

- Velocidad actual
- Velocidad objetivo
- Marcha actual
- Aceleración

Teniendo en cuenta estas variables y las restricciones, se establecen las siguientes reglas:

- Cuando la aceleración sea nula o negativa (se está decelerando), se establece modo Drive.
- Si la aceleración es positiva se pasa a modo Manual:
    - Si se puede llegar a la velocidad objetivo con la marcha actual, se mantendrá, y cuando se alcance el objetivo se establecerá modo Drive.
    - Si no se puede llegar con la marcha actual, se sube de marcha e igual que en el apartado anterior, al llegar a la velocidad objetivo, se establece modo Drive.

Capítulo 5º
Resultados

En este capítulo se muestra el resultado final de la tarjeta de circuito impreso totalmente ensamblada y conectada al vehículo y se exponen los resultados obtenidos durante las pruebas de campo.

A continuación pueden verse varias imágenes del hardware del sistema de control desarrollado totalmente montado y su integración en el vehículo.

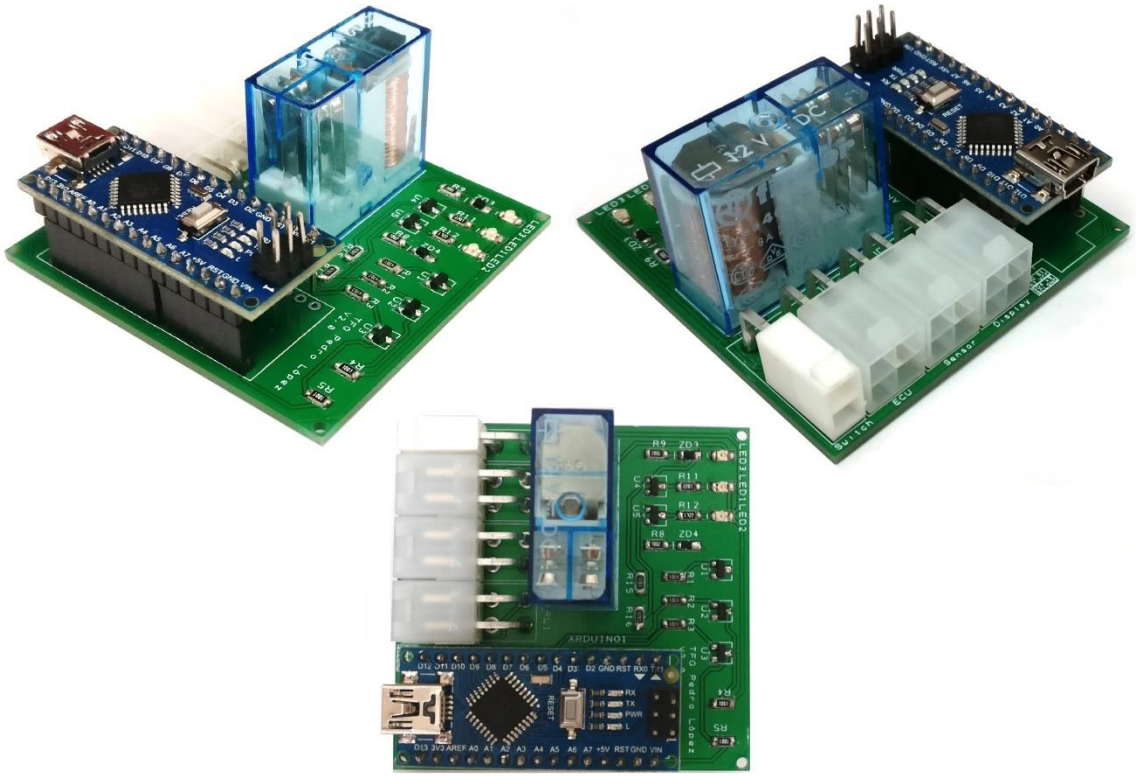


Figura 20: Hardware del sistema de control

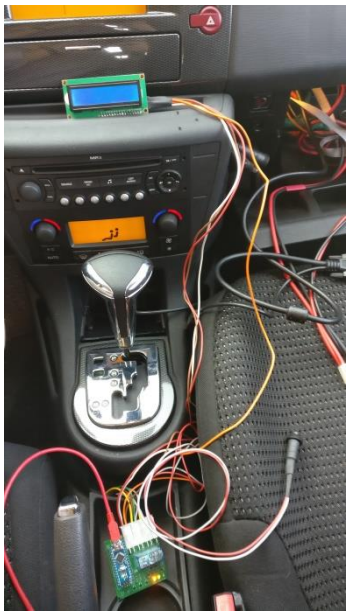


Figura 21: Montaje en el vehículo

En la figura 21 se puede ver el sistema de control preparado para hacer varias pruebas de campo. En la parte de debajo de la imagen, se observa la placa de circuito impreso conectada a la ECU, al sensor de la palanca de cambios y al PC. A la derecha, sobre el asiento se observa el interruptor que permite desconectar la alimentación de la bobina. También se puede ver la pantalla LCD que muestra el estado del sistema.

Tras realizar la conexión al vehículo, se comprobó que las señales se enviaban correctamente y que el vehículo entendía estas señales. Además, se comprobó el correcto funcionamiento del circuito encargado de la visualización y de la pantalla LCD. Para ello, con el vehículo parado, se enviaron los comandos necesarios a través de la consola de puerto serie de un PC. En las siguientes imágenes se pueden ver las distintas configuraciones.

Figura 22: Leds y LCD en modo ECU

En la figura 22 se observan las señales luminosas indicando que el relé está en posición de reposo (led rojo apagado), es decir, es el sensor de la palanca el que envía las señales a la ECU. Como los otros dos leds están encendidos, sabemos que la palanca de cambios esta en modo Drive. En la pantalla LCD se muestra la misma información. Se puede comprobar en figura 23, que el marcador del vehículo muestra esta información.

Figura 23: Marcador del vehículo en modo Drive

En la figura 24, se puede ver que los leds indican modo Drive, y además que el relé esta conmutado (led rojo encendido). El LCD muestra esa información. En cuanto al velocímetro del vehículo, muestra la misma información que la figura 23.

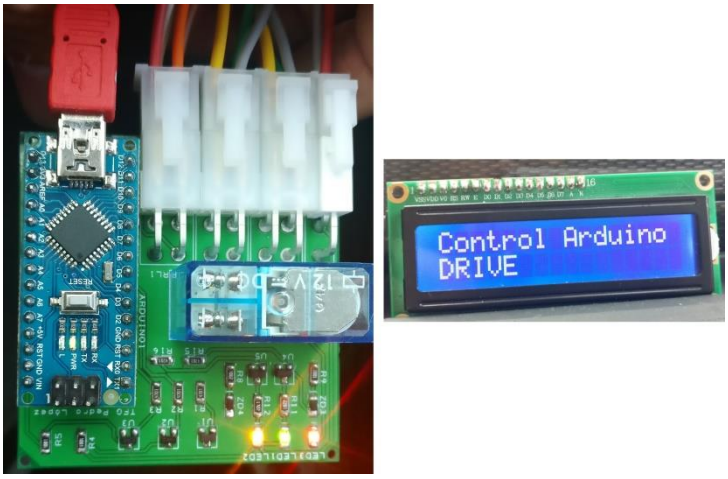


Figura 24: Leds y LCD en modo Drive

Las figuras 25, 26, 27, 28 y 29 corresponden al modo Manual, en el cual se puede subir y bajar de marcha enviando pulsos de 50ms en las señales como se explica en apartados anteriores.

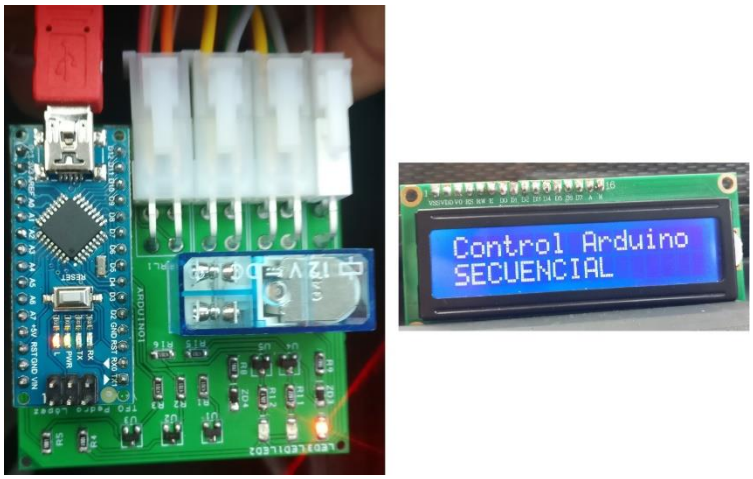


Figura 25: Leds y LCD en modo Manual

Figura 26: Marcha 1

Figura 27: Marcha 2

Figura 28: Marcha 3

Figura 29: Marcha 4

Una vez que se ha comprobado el correcto funcionamiento con el vehículo parado, se hicieron pruebas con el vehículo en marcha. Para ello, una persona iba en el asiento del conductor controlando la dirección y los pedales. Desde la posición del copiloto, se han enviado desde un PC conectado al sistema de control de marchas, los comandos necesarios para realizar una prueba completa. La prueba que se hizo comenzó con modo Drive enviado a través de la palanca de cambios. En un momento dado, se dio la orden para pasar a enviar las señales desde el sistema de control. Como el modo era el mismo, no hubo ningún cambio. A continuación se cambió a modo Manual, y en este modo se comprobó que se subía y baja marcha sin problema. Después se desconectó el relé para que fuese otra vez la palanca de cambios la encargada de enviar las señales a la ECU.

Por último se hicieron una serie de pruebas con el vehículo circulando de manera semiautónoma, es decir, el PC de control era el encargado de seguir una velocidad de referencia mientras que una persona era la encargada de controlar la dirección. Con este escenario, se hicieron 3 pruebas para ver la influencia del sistema de control de cambios sobre el comportamiento del vehículo:

- La ECU decidiendo en que momento hacer el cambio.
- Cambiando manualmente, a través de un PC conectado al sistema de control de cambios, en el momento que se ha dejado de acelerar.
- El PC del control gestionando el cambio mediante el sistema de control diseñado y el algoritmo desarrollado en el apartado 4.5.

La figura 30 corresponde al primero de los caso, la ECU decidiendo el momento del cambio. Se observa, como en la segunda rampa de subida, decide hacer un cambio de marcha de 3ª a 2ª, lo que conlleva una desviación en la velocidad real del vehículo. Para una visualización más clara, el valor de la marcha se ha multiplicado por 10.

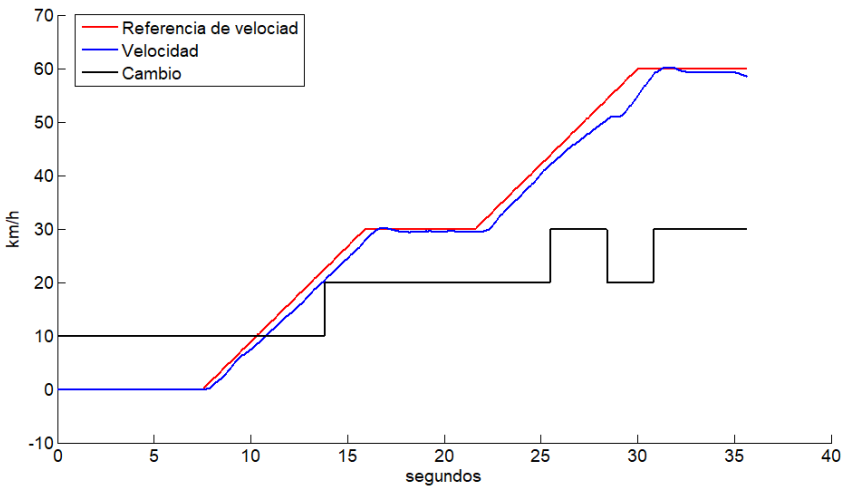


Figura 30: Velocidad del vehículo. Control de marchas por ECU

En la figura 31,al hacerse los cambios en modo Manual desde un PC , se observa que la desviación en la segunda rampa de aceleración ha desaparecido. Esto es debido a que no se ha cambiado de marcha. Al igual que en la 30, la marchas se ha multipilado por 10 para una mejor visualización.

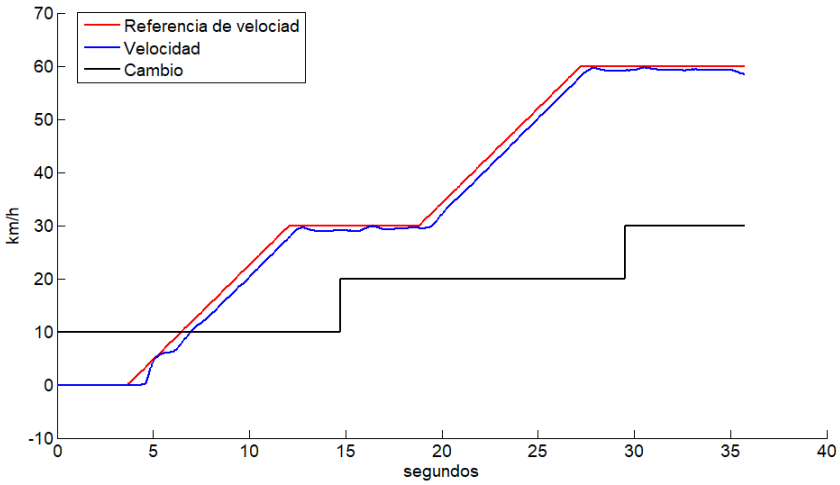
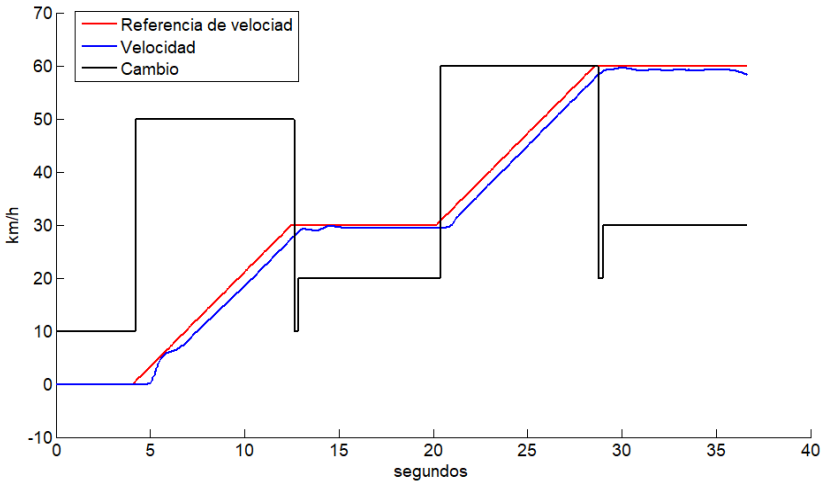


Figura 31: Velocidad del vehículo. Control manual a través del sistema diseñado

Por último, se le ha dado el control del sistema de cambios al PC de control del coche. Este ejecuta el algoritmo desarrollado en el apartado 4.5 junto con el resto de algoritmos necesarios para el control del vehículo. En la figura 32 puede verse que la desviación en la velocidad desaparece al no haber cambios de marcha en las aceleraciones. En este caso, los cambios también están multiplicados por 10, pero al haber modo Manual y Drive. En la Tabla 3 se explica la conversión.



10	DRIVE 1ª
20	DRIVE 2ª
30	DRIVE 3ª
40	DRIVE 4ª
50	MANUAL 1ª
60	MANUAL 2ª
70	MANUAL 3ª

Tabla 3: Correspondencia en las marchas

Figura 32: Velocidad del vehículo. Ejecución del algoritmo

Capítulo 6º
Conclusiones y trabajos futuros

En este capítulo se exponen las principales conclusiones extraídas durante el desarrollo de este TFG. Además se incluyen posibles mejoras que se pueden hacer en el sistema.

6.1. Conclusiones

Durante la realización de este TFG se ha extraído las siguientes conclusiones:

- Se ha diseñado e implementado un sistema de control para la caja de cambios de un coche autónomo que nos permite controlar desde un PC la marcha utilizada. Aunque debido a la naturaleza mecánica de algunas de las posiciones de la palanca de cambios no se han podido automatizar las marchas de Parking y Reversa, el sistema es capaz de controlar todas las marchas que se requieren para la conducción.
- El diseño del sistema de control ha abarcado desde el estudio del problema, pasando por la elección del hardware, el diseño de las placas, la implementación de la interfaz de control en la placa y el PC, hasta un algoritmo de demostración que mejora el funcionamiento del sistema automático del coche.
- Se han realizado experimentos para demostrar el funcionamiento del sistema en el vehículo con resultados satisfactorios.

6.2. Trabajos futuros

Para mejorar este sistema de control se podría:

- Diseño de un algoritmo de control de la marcha más complejo que tenga en cuenta las revoluciones por minuto del motor, la velocidad y aceleración objetivo y parámetros de confort.
- Implementar un sistema mecánico que permita acceder a las marchas de Parking y Reversa.
- Mejora del interfaz de manera que se permita que el LCD muestre la marcha actual.

V. BIBLIOGRAFÍA

[1] INVETT. http://www.isislab.es/

[2] GCDC. http://www.gcdc.net/en/

[3] XC Source. https://www.xcsource.com

[4] Arduino y su repositorio de código abierto. www.arduino.cc

[5] Arduino Nano. https://store.arduino.cc/usa/arduino-nano

[6] Arduino IDE. https://www.arduino.cc/en/Main/Software

[7] Desing Spark PCB https://www.rs-online.com/designspark/pcb-software

[8] RS http://es.rs-online.com/web/generalDisplay.html?id=aboutRS

[1] Ceballos Sierra, Fco. Javier. *C/C++. Curso de programación*. 4º Edición. 2015. RA-MA EDITORIAL. ISBN 978-8499645278.

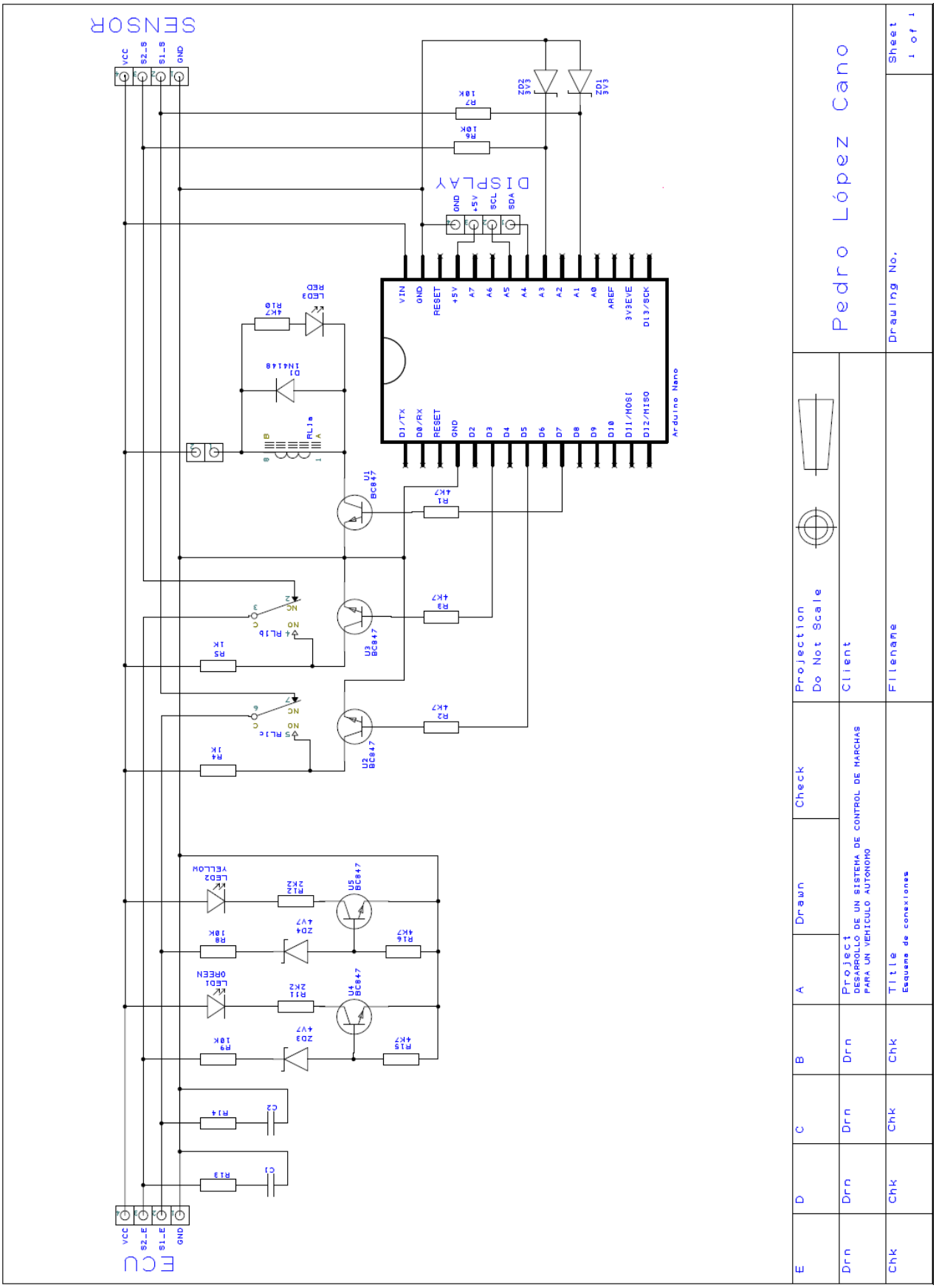
[2] Torrente Artero, Óscar. *ARDUINO. Curso práctico de formación.* 1ª Edición 2013. RC LIBROS. ISBN 978-8494072505.

[3] Blum, Jeremy. *Arduino A Fondo (Títulos Especiales).* 1ª Edición. 2014. ANAYA. ISBN 978-8441536524.

[4] López-Ferreras, Francisco, Maldonado, Santurio. *Análisis de Circuitos Lineales.*3ª Edición. RA-MA EDITORIAL. ISBN 978-8478979431.

[5] Mazo Quintas, Manuel. *Circuitos Electrónicos Digitales.* 1995. Universidad de Alcalá de Henares. Servicio de publicaciones.

VI. Anexo I. Esquema de conexiones


SENSOR
VCC
S2_S
S1_S
GND
DISPLAY
GND
+5V
SCL
SDA
Arduino Nano
VIN
RESET
D1/TX
D0/RX
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11/MOSI
D12/MISO
D13/SCK
3V3EVE
AREF
A0
A1
A2
A3
A4
A5
A6
A7
R7 10K
R8 10K
ZD2 5V3
ZD1 5V3
LED3 RED
R10 4K7
D1 1N4148
RL1a
U1 BC847
R1 4K7
U3 BC847
R3 4K7
U2 BC847
R2 4K7
R5 1K
R4 1K
RL1b
RL1c
U5 BC847
R12 2K2
LED2 YELLOW
ZD4 4V7
R16 4K7
R8 10K
U4 BC847
R11 2K2
LED1 GREEN
ZD3 4V7
R15 4K7
R9 10K
R14
C2
R13
C1
ECU
VCC
S2_E
S1_E
GND
Pedro López Cano
Drawing No.
Sheet 1 of 1
Projection Do Not Scale
Client
Filename
Check
Drawn
Project DESARROLLO DE UN SISTEMA DE CONTROL DE MARCHAS PARA UN VEHICULO AUTONOMO
Title Esquema de conexiones
A
B
C
D
E
Drn
Chk

VII. Anexo II. Código fuente del software de sistema de control

```
#include <TimerOne.h>
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#define I2C_ADDR 0x27
LiquidCrystal_I2C lcd(I2C_ADDR, 2, 1, 0, 4, 5, 6, 7);

#define ID_ARDUINO "p"
#define MAX_TASKS 10 //maximo numero de tareas
#define MS 25 // 25 Milisegundos de la interrupción
#define MS_50 2// 50 milisegundos (2 x 25ms de interrupcion)
#define SEG_2 80  //80ticks * 25 ms

#define GEAR_CONTROL_ECU 1
#define GEAR_CONTROL_MCU_DRV 2
#define GEAR_CONTROL_MCU_SEQ 3
int GEAR_CONTROL = GEAR_CONTROL_ECU;

#define GEAR_STATUS_N  1
#define GEAR_STATUS_U  2
#define GEAR_STATUS_D  3
int GEAR_STATUS = GEAR_STATUS_N;

bool lcd_refresh = false;

int S1 = 5;     //Señal cable amarillo
int S1_ECU_READ = A1;
bool S1_ECU_LAST_VAL;

int S2 = 3;     //Señal cable verde
int S2_ECU_READ = A3;
bool S2_ECU_LAST_VAL;

int rele = 7;
char puerto_serie = 0;
char recibido[11];

int LED = 13;
int LED_status = HIGH;

unsigned long int current_ticks = 0;
typedef struct {
  unsigned int ticks;
  unsigned int (*task) (void);
} Task_t;

Task_t tasks[MAX_TASKS];

void setup()
{
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(rele, OUTPUT);
  pinMode(LED, OUTPUT);

  digitalWrite(rele, LOW); //rele apagado
  digitalWrite(S1, LOW); //drive
  digitalWrite(S2, LOW); //drive
  digitalWrite(LED, LOW);


  pinMode(S1_ECU_READ, INPUT);
  pinMode(S2_ECU_READ, INPUT);
```

```
  S1_ECU_LAST_VAL = digitalRead(S1_ECU_READ);
  S2_ECU_LAST_VAL = digitalRead(S2_ECU_READ);

  //Inicializar puerto serie 9600 baudios
  Serial.begin(9600);

  // Inicializar el display con 16 caraceres 2 lineas
  lcd.begin (16, 2);
  lcd.setBacklightPin(3, POSITIVE);
  lcd.setBacklight(HIGH);

  int  estate = 0;
  while (estate != 3)
  {
    bool tr01 = 0;
    if (estate == 0 && Serial.available())
      tr01 = 1;
    bool tr12 = 0;
    if ((estate == 1) && !strcmp(recibido, ID_ARDUINO))
      tr12 = 1;
    bool tr23 = 0;
    if (estate == 2)
      tr23 = 1;
    bool tr14 = 0;
    if (estate == 1 && strcmp(recibido, ID_ARDUINO))
      tr14 = 1;
    bool tr40 = 0;
    if (estate == 4)
      tr40 = 1;

    if (estate == 0 && tr01)
      estate = 1;
    if (estate == 1 && tr12)
      estate = 2;
    if (estate == 2 && tr23)
      estate = 3;
    if (estate == 1 && tr14)
      estate = 4;
    if (estate == 4 && tr40)
      estate = 0;
    if (estate == 1)
    {
      int i = 0;
      do
      {
        recibido[i] = Serial.read();
        delay(20);
        i++;
      } while (Serial.available() && i < 10);
      recibido[i] = '\0';
    }
    if (estate == 2 || estate == 4)
    {
      Serial.print("Mi ID: ");
      Serial.println(ID_ARDUINO);
    }
  }
  deleteTasks();
  Timer1.initialize(MS * 1000); //(MS * 1000 us);
  Timer1.attachInterrupt(ISR_System);
  addTask(0, control_ECU);
  Pantalla("Control ECU", "Modo Automatico");
}
```

```
void loop()
{
  //si hay algo en el puerto serie lo guardamos
 if (Serial.available())
  {
    int i = 0;
    do
    {
      recibido[i] = Serial.read();
      delay(10);
      i++;
    } while (Serial.available() && i < 11);
    recibido[i] = '\0';
  }

  if (!strcmp(recibido, "a"))
    addTask(0, control_ECU);

  if (!strcmp(recibido, "d"))
    addTask(0, control_MCU_DRV);

  if (!strcmp(recibido, "s"))
    addTask(0, control_MCU_SEQ);

  if (!strcmp(recibido, "+"))
    addTask(0, gear_up);

  if (!strcmp(recibido, "-"))
    addTask(0, gear_down);


  bool S1_ECU_VAL = digitalRead(S1_ECU_READ);
  bool S2_ECU_VAL = digitalRead(S2_ECU_READ);
  if(S1_ECU_VAL != S1_ECU_LAST_VAL || S2_ECU_VAL != S2_ECU_LAST_VAL)
    ;//addTask(0, control_ECU);
  else
  {
    S1_ECU_LAST_VAL = S1_ECU_VAL;
    S2_ECU_LAST_VAL = S2_ECU_VAL;
  }

  if (lcd_refresh)
  {
    switch (GEAR_CONTROL)
    {
      case GEAR_CONTROL_ECU:
        Serial.println("GEAR_CONTROL_ECU");
        Pantalla("Control ECU","");
        break;
      case GEAR_CONTROL_MCU_DRV:
        Serial.println("GEAR_CONTROL_MCU_DRV");
        Pantalla("Control Arduino", "DRIVE");
        break;
      case GEAR_CONTROL_MCU_SEQ:
        Serial.println("GEAR_CONTROL_MCU_SEQ");
        Pantalla("Control Arduino", "SECUENCIAL");
        break;
    }
    lcd_refresh = false;
  }
```

```
  //borramos lo que hay en el puerto serie
  for (int i = 0; i < 11; i++)
  {
    recibido[i] = '\0';
  }
}
void ISR_System(void)
{
  //recorremos el array de tareas
  for (int i = 0; i < MAX_TASKS; i++)
  {
    //si hay algo en el tick actual
    if (tasks[i].ticks == current_ticks)
      //miramos si hay alguna tarea
      if (tasks[i].task)
      {
        //ejecutamos la tarea
        int repeticion = tasks[i].task();
        //si ha devuelto un valor de repetición
        if (repeticion > 0)
        {
          //la programamos para el nuevo instante
          tasks[i].ticks += repeticion;
        }
        else //si no hay repeticion eliminiamos la tarea
          tasks[i].task = 0;
      }
  }
  current_ticks++;

  digitalWrite(LED, LED_status);
  LED_status = !LED_status;
}
// funcion que limpia todas las tareas
void deleteTasks()
{
  for (int i = 0; i < MAX_TASKS; i++)
    tasks[i] = {0, 0};
}
//funcion que añade tareas
int addTask(unsigned int t, unsigned int (*p)(void))
{
  //recorremos el array
  for (int i = 0; i < MAX_TASKS; i++)
  {
    //si ya hay tarea en esa posición, saltamos a la siguiente
    if (tasks[i].task)
      continue;
    //si la posición está vacía
    else
    {
      //Cargamos los ticks de ejecución
      tasks[i].ticks = current_ticks + t;
      //Cargamos la tarea
      tasks[i].task = p;
      //otra manera de hacerlo sería   tasks[i] = Task_t {current_ticks+t, p};
      //no devolver error porque hemos añadido la tarea
      return 0;
    }
  }
  //devolver error si no hemos podido añadir la tarea
  return -1;
}
```

```
//función que elimina tareas
int deleteTask(unsigned int (*p)(void))
{
  for (int i = 0; i < MAX_TASKS; i++) {
    if (tasks[i].task == p)
    {
      tasks[i].ticks = 0;
      tasks[i].task = 0;
      //tasks[i] = {0, 0}; //tareaVacia();
      return 0;
    }
  }
}

int control_ECU()
{
  lcd_refresh = true;
  //estamos en cambio automatico y podemos llegar desde cualquier estado
  digitalWrite(rele, LOW);
  digitalWrite(S1, LOW);
  digitalWrite(S2, LOW);
  GEAR_CONTROL = GEAR_CONTROL_ECU;
  return 0;
}


int control_MCU_DRV()
{
  if (GEAR_STATUS == GEAR_STATUS_N && GEAR_CONTROL != GEAR_CONTROL_MCU_DRV)
  {
    lcd_refresh = true;
    digitalWrite(rele, HIGH);
    digitalWrite(S1, LOW); //coche
    digitalWrite(S2, LOW); //coche
    GEAR_CONTROL = GEAR_CONTROL_MCU_DRV;
    return 0;
  }
  else
    return -1; // return -1 borra la tarea el ISR
}


int control_MCU_SEQ()
{
  if (GEAR_STATUS == GEAR_STATUS_N && GEAR_CONTROL != GEAR_CONTROL_MCU_SEQ)
  {
    lcd_refresh = true;
    digitalWrite(rele, HIGH);
    digitalWrite(S1, HIGH); //coche
    digitalWrite(S2, HIGH); //coche
    GEAR_CONTROL = GEAR_CONTROL_MCU_SEQ;
    return 0;
  }
  else
    return -1;

}
```

```
int gear_up()
{
  if (GEAR_CONTROL == GEAR_CONTROL_MCU_SEQ)
  {
    if (GEAR_STATUS == GEAR_STATUS_N)
    {
      digitalWrite(S1, HIGH);
      digitalWrite(S2, LOW);
      GEAR_STATUS = GEAR_STATUS_U;
      return MS_50;
    }
    if (GEAR_STATUS == GEAR_STATUS_U)
    {
      digitalWrite(S1, HIGH);
      digitalWrite(S2, HIGH);
      GEAR_STATUS = GEAR_STATUS_N;
      return 0;
    }
  }
  else
    return -1;
}

int gear_down()
{
  if (GEAR_CONTROL == GEAR_CONTROL_MCU_SEQ)
  {
    if (GEAR_STATUS == GEAR_STATUS_N)
    {
      digitalWrite(S1, LOW);
      digitalWrite(S2, HIGH);
      GEAR_STATUS = GEAR_STATUS_D;
      return MS_50;
    }
    if (GEAR_STATUS == GEAR_STATUS_D)
    {
      digitalWrite(S1, HIGH);
      digitalWrite(S2, HIGH);
      GEAR_STATUS = GEAR_STATUS_N;
      return 0;
    }
  }
  else
    return -1;
}


int Pantalla (char a[16], char b[16])
{ //funcion que pone en el display algo
  //el primer dato es la primera linea
  //el segundo la segunda linea

  lcd.home();
  lcd.clear();
  lcd.write(a);
  lcd.setCursor (0, 1);
  lcd.write(b);
}
```

VIII. Anexo III. Código fuente del algoritmo del sistema de cambio de marcha

```
#define MAX_SPEED_GEAR1 40
#define MIN_SPEED_GEAR2 20
#define MAX_SPEED_GEAR2 80
#define MIN_SPEED_GEAR3 40
#define MAX_SPEED_GEAR3 100
#define MIN_SPEED_GEAR4 60
#define MAX_SPEED_GEAR4 120
#define MODE_MCU_DRV    0
#define MODE_MCU_SEQ    1

void gearControl(unsigned int gear, float current_speed, float target_speed,
float acceleration, int *mode, unsigned int *desiredGear)
{
  *desiredGear = gear;

  //si deceleración o aceleración 0
  if (acceleration <= 0)
    *mode = MODE_MCU_DRV;
  //si aceleración positiva
  else
  {
    //modo Secuencial
    *mode = MODE_MCU_SEQ;
    //evaluamos la marcha actual
    switch (gear)
    {
      case 1:
        //si velocidad objetivo en el rago de 1ª, no se hace nada
        if (target_speed < MAX_SPEED_GEAR1)
        {
          *desiredGear = 1;
          break;
        }
        //si vel. objetivo y vel. actual en el rango de 2ª se sube marcha
        if (target_speed < MAX_SPEED_GEAR2)
        {
          *desiredGear = 2;
          break;
        }
        else
        {
          *mode = MODE_MCU_DRV;
          break;
        }
      case 2:
        //si velocidad objetivo en el rago de 2ª, no se hace nada
        if (target_speed < MAX_SPEED_GEAR2)
        {
          *desiredGear = 2;
          break;
        }

        //si vel. objetivo y vel. actual en el rango de 3ª se sube marcha
        if (target_speed < MAX_SPEED_GEAR3)
        {
          *desiredGear = 3;
          break;
        }
        else
        {
          *mode = MODE_MCU_DRV;
          break;
        }
```

```
      case 3:
        //si reduciendo una vez se llega, se reduce
        if (current_speed > MIN_SPEED_GEAR2 && target_speed < MAX_SPEED_GEAR2)
        {
          *desiredGear = 2;
          break;
        }
        //si velocidad objetivo en el rango de 3ª, no se hace nada
        if (target_speed < MAX_SPEED_GEAR3)
        {
          *desiredGear = 3;
          break;
        }
        //si vel. objetivo y vel. actual en el rango de 4ª se sube
        else
        {
          *desiredGear = 4;
          break;
        }

      case 4: // en 4ª solo podemos reducir, porque no se puede subir más
        //Si reduciendo una vez se llega, se reduce
        if (current_speed > MIN_SPEED_GEAR3 && target_speed < MAX_SPEED_GEAR3)
        {
          *desiredGear = 3;
          break;
        }

    }
  }
}
```